Introduction to User-Mode Linux

How to install, configure, and use Linux virtual machines

Skill Level: Introductory

Carla Schroder (dworks@bratgrrl.com)
Programmer and author
Freelance

23 Jan 2003

This tutorial shows how to install, configure, and use Linux® virtual machines. With User-Mode Linux (UML), you can set up multiple virtual machines that are isolated from each other and from the hardware. This lets you test applications all the way to failure without breaking the host system -- or even requiring a reboot.

Section 1. Before you start

About this tutorial

Ever wish you had a place to let your Linux applications play -- where they wouldn't hurt anything else? Do your killer apps spend too much time killing each other? Originally conceived as a kernel developer's tool, UML lets you set up multiple virtual machines that are isolated from each other and from the hardware. Now, you can test applications all the way to failure without breaking the host system -- or even requiring a reboot. Veteran administrator Carla Schroder shows you how.

Prerequisites

You should be at least an intermediate Linux user: comfortable with working from the command line and building software from source; familiar with the Linux filesystem

structure; mounting devices and filesystems; managing user accounts.

Section 2. What is User-Mode Linux?

UML overview

The Linux kernel has the ability to run on top of itself in user-space. Lo and behold, instant virtual machines. Of course it's not quite that simple; the principal author Jeff Dike, and a horde of contributors have put a lot of work into UML.

It was originally designed as a kernel developer's tool, to speed up development times and reduce hardware requirements. Each instance of User-Mode Linux is safely contained and isolated from the interacting directly with hardware, and from all other instances of UML running on the same computer. This enables all sorts of test-to-destruction scenarios, without damaging the host system or neighboring VMs, and without requiring reboots when things go blooey.

UML's many uses

UML is free and complete with source code, and users are continually coming up with ever-more-imaginative ways to employ it:

- As a sandbox for testing new apps
- For testing development kernels safely
- As a way to run different Linux distributions simultaneously
- For creating various development environments, with different compilers, different libraries, and different filesystems
- For virtual networking
- For virtual Web hosting
- For virtual clusters
- As supreme chroot jails
- To emulate hardware and resources not physically present

Section 3. Installing UML

Getting started

Required are a computer running Linux, a UML kernel, and a root filesystem. The host operating system can be most any modern Linux with a 2.2.15 kernel or better. (Run the command `uname -a` to get the kernel version.) I have it on two home-built PCs: an AMD Duron 800 with 256MB of RAM and Red Hat 8.0; and an older system, a Celeron 333 with 256MB of RAM running Libranet 2.7. Not what I would characterize as brawny, but sufficient for some serious UML tinkering.

The easiest way to get started is to download and install a UML RPM or deb and a prefabricated filesystem. Both are available from the UML project page (see Resources). The packages contain the kernel, documentation, and utilities. Of course, as with all things Linux, there are endless customization options. We'll take the easy way first.

A word of caution: UML is evolving rapidly, and has some rough edges. It is mature enough to use in a production environment, but it has a bit of a learning curve. Kernel hackers will likely slip into it more comfortably than us ordinary mortals. Fear not, if an ol' country sysadmin like me can use it, likely anyone can.

RPM-based installation

There is a single generic UML kernel RPM, the current one is user_mode_linux-2.4.19.5um-0.i386.rpm (you can download this from the link in Resources). Always use the latest package; UML is in a continual state of advancement. To see what's inside, list the package files:

```
# rpm -ql user_mode_linux
```

To install, run

```
# rpm -ivh user_mode_linux-2.4.19.5um-0.i386.rpm
```

Verify installation by running /usr/bin/linux. It should boot partway, then kernel panic when it doesn't find a root filesystem. Enjoy the moment- UML lets a person panic kernels all they want, with no bad consequences.

Choose a filesystem

Prefab filesystems abound. They are compressed with bzip2, so expect a high compression ratio. For example, root_fs.rh-7.2-full.pristine.20020312.bz2 expands from 170MB to 679MB. This creates a complete Red Hat 7.2 installation. The smallest is root_fs_toms1.7.205.bz2, Tom's root boot. For those unfamiliar with Tom's root boot, or tomsrtbt as it is officially spelled, it is "the most GNU/Linux on one floppy disk," the most amazing collection of utilities crammed onto a single bootable floppy disk you ever saw.

A fun show-off stunt is to take a bare machine with absolutely nothing installed on it and, using only tomsrtbt, build a complete Linux system over FTP. User-Mode Linux is a wonderful test bed to practice using tomsrtbt; hone your skills, then amaze and impress friends on any machine.

A nice compact option is the Debian Woody download, Debian-3.0r0.ext2, it unpacks to about 60MB. It's a complete root filesystem, ready for installing development tools and other apps. One gotcha: UML needs devfs, which is not included in Woody. See the Debian installation section for what to do. (Hint: apt-get....)

Note that most UML operations on the host can be performed as a user, rather than root. This is by design, for greater security on the host system, and ease of use.

Install the filesystem

Make a UML directory, like so:

```
# mkdir /opt/uml
# chmod 755 /opt/uml
```

Why /opt? No particular reason, except it's sitting there, doing nothing. If multiple users are part of your plan, it's not a good idea to put shared things in /home directories. Instead, put them out in a public directory. `chmod 755` gives read and execute rights to everyone, and write to root only. Adjust permissions according to your needs.

Unpack your chosen prefab filesystem:

```
$ bunzip2 root_fs.rh-7.2-full.pristine.20020312.bz2
```

bunzip2 will automatically delete the compressed file, leaving only the unpacked file. See man bzip2 for complete command options.

Make a soft link to /usr/bin/linux:

```
$ ln -s /usr/bin/linux /opt/uml/linux
```

My usual practice is to create a command directory containing symlinks, and store the actual binaries and libs elsewhere. This caused some unpredictable behaviors with UML, which may be related to the particular Linux used. Libranet let me link anyway I want; Red Hat 8 would not boot UML from a soft-linked filesytem. I found the most surefire way to make things work smoothly was to boot from the directory containing the root filesystem.

```
[carla@localhost carla]$ cd /opt/uml
[carla@localhost uml]$ ls -lhs
total 680M
   0 lrwxrwxrwx  1 carla  carla    14 Dec 2 20:52 linux -> /usr/bin/linux
680M -r-xr-xr-x  1 root   carla  679M Dec 2 19:27 root_fs.rh-7.2-full.pristine.20020312
```

Most Linuxes include /usr/bin in the default path, so having the symlink may not be necessary. For simplicity, rename the filesystem `root_fs`. UML by default looks for `root_fs` in the current directory.

Section 4. Working with UML

Boot UML

To boot up a shiny new UML, all ready to go to work, type in this command:

```
[carla@localhost uml]$ ./linux
```

Passwords and image selection

The RPM filesystems come with two built-in accounts: root, with the password "root," and user, with password "user." If you are not prompted to change passwords on first use, it's a good idea to do so anyway. Add users the usual way:

```
bash-2.05# useradd alice
bash-2.05# passwd alice
Changing password for user alice
Enter new UNIX password:
Retype new UNIX password:
```

```
passwd: all authentication tokens
updated successfully
```

A collection of various root filesystem images can be assembled, selectable with a boot option. The way to choose a different image at boot is to assign the filesystem name to ubd0, using the syntax `ubd0=rootfs_filename`:

```
$ ./linux ubd0=/some/root/filesystem/image
```

Make sure you're booting UML to an image, rather than a mounted filesystem. ubd0 is UML's boot device. ubd represents configured block devices. ubd is UML's designation for hd, fd, cdrom, scd, and so forth. This is where devfs comes into play. /dev is limited and inflexible, cluttered with hundreds of entries that have nothing to do with what's really installed on the system. devfs is a virtual directory created at run time, containing only entries to block devices actually present on the system. ubd devices are extremely flexible: You can map them to partitions, directories, and volume managers, as well as hardware devices, and you can load and unload devices from a running instance of UML with the management console, mconsole.

Mounting devices

We have a nice instance of UML running, but it's rather limited. No swap, no CD, no floppy, no networking. Take a gander in /dev/udb:

```
bash-2.05# cd /dev/ubd
bash-2.05# ls
0  disc0
```

Not much going on here. Adding devices is the same in UML as in any Linux: add entries to /etc/fstab and create a directory in /mnt, or mount them manually. There is one additional UML-specific step, which is to define devices at boot:

```
$ ./linux ubd0=rootfs ubd1=swap ubd2=/dev/cdrom
```

Creating files inside UML is discussed below. Mounting devices manually inside UML uses the usual "mount" syntax, with ubd*n* defining the device name:

```
mount /dev/ubd/2 /mnt/cdrom
```

Notice that the boot options are ubd*n*, whereas the mount command options are /ubd/*n*.

Creating elbow room using sparse files

Exit UML. Use the usual Linux commands, such as `shutdown -h now`. It's time to create some dedicated elbow room for UML to run in by creating and formatting a *sparse* file. This is a nice little bit of magic that allocates a fixed amount of space for a file without actually using all of it.

```
$ dd if=/dev/zero of=uml_root count=1 bs=1k seek=$[2*1024*1024]
1+0 records in
1+0 records out
```

Let's call it uml_root instead of root_fs, as root_fs is often used in the UML docs as a variable. Variables are in italics.

Confirm file creation:

```
$ ls -lhs
total 12k
12K -rw-r--r--    1 root     root         2.0G Dec 2 12:35 uml_root
```

See the wizardry? A 2GB file that occupies 12KB of space on disk. The "s" switch shows how much space on disk a file actually occupies.

Any Linux file format will work. This formats as ext2:

```
$ /sbin/mkfs -t ext2 uml_root
```

Now comes the tricky part. There are various ways to do this, but this one works nicely. I'm not going to show the output here, just the commands. Remember, root_fs is the original UML filesystem image. This copies it into the uml_root sparse file:

```
# mkdir /mnt/m1
# mkdir /mnt/m2
# mount -o loop -t ext2  /opt/uml/root_fs  /mnt/m1
# mount -o loop -t ext2 uml_root  /mnt/m2
# cp -a /mnt/m1/* /mnt/m2/
```

Look, filesystems! Run ls -lhs on /mnt/m1 and /m2 to compare. Now unmount both:

```
# umount /mnt/m1
# umount /mnt/m2
```

If you get a "device busy" error, close out everything that may be reading the filesystem, such as file managers and xterms, and try again. Now boot up UML in the normal fashion, pointing to your shiny new filesystem:

```
[carla@localhost uml]$ ./linux ubd0=uml_root
```

Feel the power? Aside from its usefulness, UML is just plain fun.

Creating a filesystem inside UML

The filesystem can also be created inside UML, which is geekier and more fun. First, create the sparse file on the host; I called it uml_root2. Then boot up a UML session:

```
$ ./linux ubd0=uml_root  ubd2=/opt/umltest/uml_root2
```

As root, run these commands:

```
# /sbin/mkfs -t ext2 /dev/ubd/2
# mount /dev/ubd/2 /mnt
# cp -ax / /mnt
```

Make sure to use an unassigned device; ubd1 is typically assigned to swap. Kick back and take a walk. If your filesystem is a large one, the last step will hammer your CPU for a few minutes. When it's all done, shut down then start up with the new filesystem:

```
$ cd /opt/umltest/
$ linux ubd0=uml_root2
```

File creation chores are not done yet, there is no actual swap partition, so we need to make one:

```
# dd if=/dev/zero/ of=swapfs bs=1k count=1 seek=$[512*1024]
# /sbin/mkswap swapfs
```

On boot, be sure to specify ubd1=swapfs. It may also be necessary to run the swapon command inside the UML session:

```
# /sbin/swapon -a
```

All file creation chores can be done inside UML. This eliminates the need for root

privileges on the host and lets users manage UML sessions just as though they were on a stand-alone system. The main reason for creating something like a 2GB sparse file on the host is to limit disk space for your users.

Section 5. Working with multiple UMLs

Have a COW, man

Playing with a single UML is okay, but the real fun comes with running a lot of UMLs. One option is to install many separate, individual filesystems. Another option is Copy-On-Write, or COW. All users share the same root filesystem, only data and user modifications to the root filesystem are kept separate. One user can create many COW files, and run them simultaneously. This is not suitable for testing different Linux distributions, or for users who make extensive modifications to their root filesystems. It's perfect for the ace kernel hacker, virtual Web hosting, sandboxes for application testing, or any environment where multiple users don't need major modifications to the root filesystem.

Creating a COW filesystem is simple. Set the root filesystem to write-only:

```
[root@localhost uml] # chmod 444
uml_root
```

Then boot with this option:

```
[carla@localhost uml] $ ./linux ubd0=/home/carla/uml_root.cow,uml_root
```

Mind your spaces: there are none on either side of the comma. This creates a .cow file for user carla on the host system, in carla's home directory. Now the original uml_root file will not be written to. This is called the "backing" file. uml_root.cow is where changes for user carla are recorded. Name the COW files anything you like. Subsequent logins can be made from the user's home directory:

```
[carla@localhost carla] $ ./linux ubd0=uml_root.cow
```

Let's see how much disk space we are saving:

```
[carla@localhost uml]$ ls -lhs
total 680M
lrwxrwxrwx    1 root     carla         14 Dec  2 20:52 linux -> /usr/bin/linux
-r-xr-xr-x    1 root     carla        679M Dec  2 19:27 uml_root

[carla@localhost carla]$ ls -lhs
total 2.9M
2.9M -rw-r--r--    1 carla    carla        641M Dec  2 22:27 uml_root.cow
```

677.1MB saved. Even with today's large hard drives, that's a tidy chunk. It is important that the original backing filesystem be protected; if anything changes, UML will not mount the device. This means it is not possible to upgrade the COW files by upgrading the backing system. It is possible to merge a COW file with the backing file, using the uml_moo utility:

```
# uml_moo  cow_file  new_backing_file
```

Keep a copy of the original backing file in case this gives bad results. COW files can be stored anywhere your heart desires on the host system:

```
$ ./linux ubd0=/really/odd/file/location/root_fs.cow,root_fs
```

COW files are great for serious testing-to-destruction. Rather than wasting one's life fscking, or waiting for reboots, simply delete and re-create COW files as needed.

mconsole

Give each UML instance a unique ID by adding this to the boot options:

```
umid=foo
```

For example, when I have a lot of UML sessions running, I name them umid=carla1, umid=carla2, and so forth. This gives quick access to the management console, which lets the user do things to a running UML session from the host:

```
$ uml_mconsole carla1 command
```

Look in your home directory on the host system. There's a .uml directory full of files with odd, apparently random-character filenames. mconsole is always activated at boot. If the user does not specify a umid, it is given one of those wacky random filenames:

```
mconsole (version 2) initialized on /home/carla/.uml/HUrvzP/mconsole
```

mconsole commands

Obviously, it's easier to find one given a sensible name. mconsole has a limited, but useful set of commands. My favorite is "halt":

```
$ uml_mconsole carla1 halt
```

It's a drastic halt, akin to pulling the plug. Sometimes that's what it takes. "reboot" is just as drastic.

- "help" displays the commands
- "config" adds devices to the VM on the fly, using the same syntax as the boot command line: config eth0=tuntap, config eth1=mcast
- "remove" deletes devices, using the name of the device. For example: remove ubd2
- "sysrq" calls the kernel's SysRq driver. See documentation/sysrq.txt in your kernel tree for command options. This is kernel hacker-land; other users mess with this at their peril. Well no, it's inside UML, no worries, have at it.

Section 6. Networking

Transports

The prefab kernels have virtual networking enabled. For custom-built kernels, enable "Network device support" and the three transports. There are several networking transports:

- ethertap
- TUN/TAP
- Multicast
- switch daemon

- slip
- slirp
- pcap

We'll look at TUN/TAP, ethertap, and multicast. TUN/TAP allows exchanging packets between the host and the virtual machines with 2.4 kernels. Multicast is purely virtual, for networking the VMs only. Ethertap is for host access on 2.2 and 2.4 kernels, though officially it is obsolete on 2.4. It works fine on 2.4; TUN/TAP gives better performance and security.

For both ethertap and TUN/TAP, two IPs are needed: one for the host, one for the UML. The host IP acts as a gateway. If the host already has an IP, use it, or make up a totally new one as you see fit. The kernel boot command looks likes this:

```
eth <n> = <transport> , <device> , <ethernet address> , <tap IP address>
```

Ethertap

Let's give our host the IP of 192.168.1.100. Add to the UML kernel boot options:

```
eth0=ethertap,tap0,fe:fd:0:0:0:1,192.168.1.100
```

fe:fd:0:0:0:1 is the Ethernet address assigned to the UML eth0, 192.168.1.100 is the IP. An easier way is to omit fe:fd:0:0:0:1, and let the uml_net helper assign one. This resides in /user/bin/uml_net, so it should be in your path and automatically available. Don't try to run it manually. Warning: uml_net needs to run as root, so there are potential security problems. It's the easiest way to get networking up and running, so please see the UML docs for more advanced options.

```
eth0=ethertap,tap0,,192.168.1.100
```

That's the easy way, where the uml_net helper takes care of the host setup. Notice the commas; all of them are needed. After logging in to UML, run ifconfig:

```
# ifconfig eth0 192.168.1.101 up
```

Add a route to the host:

```
# route add default gw 192.168.0.1
```

TUN/TAP

First, see if the tun.o module is loaded in the kernel on the host system. The pre-fab kernels will have it:

```
# /sbin/lsmod
```

A long list should appear. The left-hand column says Modules. Look for:

```
Module                  Size  Used by    Not tainted
tun                     5696   0  (unused)
```

If it's not there, use insmod to load it:

```
# /sbin/insmod  /filepath/tun.o
```

Same UML kernel boot command syntax as ethertap, with fewer arguments:

```
eth0=tuntap,,,192.168.1.100
```

Boot UML, run ifconfig and route just like with ethertap.

Two potential TUN/TAP problems to look out for:

- TUN/TAP seems not to work on 2.4.3 and earlier. Upgrade the host kernel or use ethertap.
- With an upgraded kernel, TUN/TAP may fail with "File descriptor in bad state." This is due to a header mismatch between the upgraded kernel and the kernel that was originally installed on the machine. The fix is to make sure that /usr/src/linux points to the headers for the running kernel.

A quick review of TUN/TAP: TUN is a virtual point-to-point network driver, providing low-level kernel support for Ethernet tunneling. TAP is the virtual Ethernet device. TUN speaks IP, TAP speaks Ethernet, and TUN/TAP supports bridging. Lots of versatility in a small package.

Multicast

Networking the VMs is as easy as eating pie. This requires multicast in the host kernel. Most likely it is there, but if not, enable "IP: multicasting" during kernel compilation. Also needed is a multicast-capable network device on the host, like an Ethernet NIC, and eth0. On every VM, add this boot option:

```
eth0=mcast
```

Log in, then configure eth0 in the usual way on each VM:

```
# ifconfig eth0 192.168.1.112
# ifconfig eth0 192.168.1.113
# ifconfig eth0 192.168.1.114
```

Available multicast parameters are: ethX=mcast,hwaddr,mcastgroup,port,ttl. mcastgroup has a default of 239.192.168.1, the proper multi-cast address for local organizational use. The default port is 1102, time to live is 1. Multicast TTL of 1 means packets will never leave the local network segment. Change this to higher numbers as needed, for example when adding routers to your multicast network.

Troubles

There are a number of silly little gotchas that bite even the best network guru:

- Make sure kernel modules are loaded.
- Mind spaces and cases.
- The host and the virtual machine need different IPs, on the same subnet, with the same netmask, unless, of course, you set up routing. In fact this is a dandy mechanism for practicing setting up routes.

Section 7. Advanced topics

Accessing files on the host

The host's files can be mounted and accessed. Just like anything in UNIX, everything is a file. Make sure hostfs is available on the VM:

```
```

```
$ cat /proc/filesystems
nodev hostfs
      root-hostfs
```

The prefab kernels already have it. If it's not in the kernel, recompile to include it. Run mount:

```
# mount none /mnt/host -t hostfs
```

If it complains that /mnt/host does not exist, mkdir /mnt/host and run mount again. And voila, there it is. To specify a subdirectory:

```
# mount none /mnt/home -t hostfs -o /home/carla
```

Running X

Now we come to The True Test, running X. It is possible to connect an X client directly to the host X server, but Xnest is better. Xnest is an X server that is a client of the X server on the host. (I don't know about you, but I'm starting to get that hall-of-mirrors feeling with this project.) In other words, it acts as a local server to your UML session. Download and install Xnest on the virtual machine. Debian users can simply run `apt-get install xnest`. RPM- based systems can find it on RPMFind. Xnest is simple to install and set up, so let's skip to how it works on our UML virtual machines.

Xnest must first connect to the host X server. On the host machine, run this command pointing to the VM IP and display:

```
$ xhost 192.168.1.110:0
```

On the VM:

```
$ DISPLAY=host :0 Xnest &
```

Call up your favorite window manager. I like IceWM:

```
$ DISPLAY= :0 icewm &
```

Even with the efficiencies of Xnest and lightweight window managers, running lots of X sessions sucks up system resources in a hurry. I consider this a fine reason for purchasing the latest, greatest hardware. After all, over-stressing hardware leads to

inefficiency, errors, and vexation.

Calculating system requirements

Before booting up UML, this is what `free` measured on the host system:

```
$ free
             total       used       free     shared    buffers     cached
Mem:        255408     212692      42716          0       8600     117304
-/+ buffers/cache:      86788     168620
Swap:       522072      33316     488756
```

After booting the Red Hat 7.2 UML:

```
$ free
             total       used       free     shared    buffers     cached
Mem:        255408     245528       9880          0       2844     149584
-/+ buffers/cache:      93100     162308
Swap:       522072      33308     488764
```

As you can see, it sucked up nearly 33MB of physical memory, without running any applications. This is a rather fat filesystem; by default it loads every imaginable service: squid, sshd, inetd, named, httpd, sendmail, telnet, you name it. Starting up more UML sessions, running bloatier distros, and running programs will use even more. Even though Linux manages system resources efficiently, there's no substitute for having brawny enough hardware.

Building the kernel from scratch

I won't presume to teach real programmers what to do, but here are some tips and tricks for all users. Download the latest UML patch, then download the matching kernel from a kernel mirror. Unpack the kernel. Apply the patch:

```
$ bzcat patch.bz2 | patch -p1
```

Do not mingle this with the host kernel sources in /usr/src/linux. Keep it in a separate directory. Test the patch first:

```
$ bzcat patch.bz2 | patch -p1 --dry-run
```

If you have problems with bzcat, unpack the patch first, then try cat:

```
```

```
$ cat patch.bz2 | patch -p1 --dry-run
```

Compile in the usual way, with one important addition:

```
$ make xconfig ARCH=um
$ make linux ARCH=um
```

Be sure to specify ARCH=um, since what we want is UML architecture.

UML Builder

UML Builder is a dandy utility for installing different Linux distributions into UML, using their native installation routines. This makes it possible to run and upgrade a Linux distribution just as though it were a regular, stand-alone installation. Currently, 17 distributions are supported. Ace developers who can add more are always welcome. After installing UML Builder, on the host system run this:

```
# umlbuilder
```

or

```
#umbuilder_gui
```

to get the pretty graphical version. Among other nice features, UML Builder sets up Xnest and networking.

Debian installation

I tested UML on Libranet, a marvelous user-friendly distribution built on Debian. It has the legendary Debian stability, a simplified install, and an extensive collection of desktop applications. It's not necessary to run the Debian filesystems on a Debian host, I just felt like doing it that way.

First, update the package directory database:

```
# apt-get update
```

Then run

```
# apt-get install user-mode-linux
```

The usual assortment of stable, testing, and unstable packages are available. The Libranet default in /etc/apt/sources.list is Woody (stable). UML is bleeding-edge enough all on its own, so I vote for stable. Or go to the Debian site (see the Resources section for links) to grab the packages manually.

apt-get does all the work, so sit back and relax. When it's finished, the UML kernel will be compiled and ready to run. To see what was installed on your system, see /var/lib/dpkg/info/ and look for user-mode-linux and uml-utilities.

Download a pre-fab root filesystem and away you go, just like on a RPM-based Linux. See Resources for a link to two Debian filesystems. (Of course, any filesystem will work, it doesn't have to be Debian.) The one gotcha with these is the Debian Woody kernels do not have devfs packaged with them, and UML needs devfs.

Set up networking in the VM, then:

```
# apt-get install devfsd
```

With or without devfs, it will boot up in the normal manner and fling an array of six virtual consoles onto your screen, waiting for a login:

```
Debian GNU/Linux 3.0 (none) tty1
(none) login:
```

The Debian filesystems are a little different from the others. They do not come with the two built-in "root" and "user" accounts. Just type root, hit enter, and you're in. It will not nag you to do the right thing; be sure to add a user account (`adduser`) and give root a password (`passwd`) before you do anything else. It is possible to run UML without using devfs, but that means you'll have to create and make entries in /dev/udb. /etc/fstab and /etc/inittab will have to be edited as well.

Section 8. Summary

User-Mode Linux is endlessly adaptable, but this should get you off to a good start. Please visit the Resources section and the User-Mode Linux home page for bales of additional excellent information. Visit the mailing lists and the IRC channel for help, and mingle with other UML users. Please note the Help! section on the UML front page, and take advantage of the many opportunities to make a contribution to this excellent project.

Resources

Learn

- Here are some resources you may find helpful:
    - Get more information on UML from the UML home page on SourceForge.
    - You can download Libranet, a user-friendly distribution based on Debian, from the Libranet site.
    - "Tom's floppy, which has a root filesystem and is also bootable" (Tomsrtbt) is a useful set of utilities, all on a single bootable floppy.
    - UML Builder is a tool that lets you install RPM-based Linux distributions for use with UML.
    - SourceForge provides more information on Virtual Point-to-Point (TUN) and Ethernet (TAP) devices.
    - Get more information on Xnest on xfree86.org. You can download the Xnest RPM from rpmfind.net.
    - The Linux Kernel Archives is the main site for Linux kernel source.
    - You can download the Debian UML kernel and utilities on debian.org. You'll also find two Debian filesystems.
- Find more tutorials for Linux developers in the developerWorks Linux zone.
- Stay current with developerWorks technical events and Webcasts.

Get products and technologies

- Order the SEK for Linux, a two-DVD set containing the latest IBM trial software for Linux from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.
- Download IBM trial software directly from developerWorks.

Discuss

- Read developerWorks blogs, and get involved in the developerWorks community.

About the author

Carla Schroder

Carla Schroder is a freelance PC tamer, administering Linux and

Windows systems for small businesses. She writes how-tos for real people, loves computers and high tech, and thinks Linux/Open Source/Free Software is the best playground in the world. Carla discovered computers and high-tech in 1994, her first PC was an Apple II. She progressed through DOS/Windows, from 3.1 to XP. She discovered Linux in 1998. You can contact Carla at dworks@bratgrrl.com.